CS4311 Design and Analysis of Algorithms

Homework 5 (Solution Sketch)

1. **Main Idea:** The camel can always pick the *farthest* oasis that is reachable as the next stop for supply.

   The Greedy Choice property can be proven by "Cut-and-Paste" argument.
   The Optimal Substructure property can be easily proven by contradiction.

2. **Main Idea:** To ease the discussion, for any integer i, we define $\bar{i}$ to be the largest power of two that is smaller than or equal to i.

   (a) Applying the aggregate method, we see that the total cost for pushing is $O(n)$, and the total cost for flipping is at most $1 + 2 + 4 + ... + \bar{n} = O(n)$. Thus, the amortized cost is $O(1)$ for each operation.

   (b) For accounting method, we assign \$3 for each operation. Suppose the current number of items is i. By induction, we can show that for each item added after the $\bar{i}$th operation, they will possess \$2, while for all the others each possesses at least \$0. Thus, whenever there is a flipping event, the money in the stack is enough to pay for the cost. Thus, \$3 is enough, and the amortized cost is $O(1)$.

   (c) The potential function can be designed based on the accounting method. We assign the potential function Φ to be $2(i - \bar{i})$. It can be shown that no matter flipping occurs or not, the amortized cost of each operation is at most \$3, which is $O(1)$.

3. **Main Idea:** For each node v in the heap, let $f(v)$ denote the number of nodes in the subtree rooted at v, and let $d(v)$ denote the node-depth of v.

   The following two potential functions both can show that the amortized cost of Insert is $O(\log n)$, and the amortized cost of Extract-Min is $O(1)$.

   (a) $\Phi = \sum_v d(v)$, or

   (b) $\Phi = \sum_v f(v)$.

   However, it is impossible to have the amortized cost of Insert to be $o(\log n)$ and at the same time the amortized cost of Extract-Min is $O(1)$. Assume on the contrary that this can be done. Then, we can sort n items using n Insert followed by n Extract-Min, with a total cost of $n \times o(\log n) + n \times O(1) = o(n \log n)$. As each operation of Insert or Extract-Min requires only a series of comparisons, this shows that n items can always be sorted using $o(n \log n)$ comparisons. Now, we obtain a contradiction from the sorting lower bound, which states that sorting n items in the worst case needs $\Omega(n \log n)$ comparisons.